Graph-Controlled Insertion-Deletion Systems

Rudolf Freund

Faculty of Informatics, Vienna University of Technology, Favoritenstr. 9, 1040 Vienna, Austria

`rudi@emcc.at`

Marian Kogler

Faculty of Informatics, Vienna University of Technology, Favoritenstr. 9, 1040 Vienna, Austria
Institute of Computer Science, Martin Luther University Halle-Wittenberg
Von-Seckendorff-Platz 1, 06120 Halle (Saale), Germany
`marian@emcc.at, kogler@informatik.uni-halle.de`

Yurii Rogozhin

Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Str. Academiei 5, Chişinău, MD-2028, Moldova

`rogozhin@math.md`

Sergey Verlan

LACL, Département Informatique, UFR Sciences et Technologie
Université Paris Est, 61, av. Général de Gaulle, 94010 Créteil, France

`verlan@univ-paris12.fr`

In this article, we consider the operations of insertion and deletion working in a graph-controlled manner. We show that like in the case of context-free productions, the computational power is strictly increased when using a control graph: computational completeness can be obtained by systems with insertion or deletion rules involving at most two symbols in a contextual or in a context-free manner and with the control graph having only four nodes.

1   Introduction

The operations of insertion and deletion were first considered with a linguistic motivation [14, 4, 17]. Another inspiration for these operations comes from the fact that the insertion operation and its iterated variants are generalized versions of Kleene's operations of concatenation and closure [10], while the deletion operation generalizes the quotient operation. A study of properties of the corresponding operations may be found in [6, 7, 9]. Insertion and deletion also have interesting biological motivations, e.g., they correspond to a mismatched annealing of DNA sequences; these operations are also present in the evolution processes in the form of point mutations as well as in RNA editing, see the discussions in [1, 2, 21] and [19]. These biological motivations of insertion-deletion operations led to their study in the framework of molecular computing, see, for example, [3, 8, 19, 22].

In general, an insertion operation means adding a substring to a given string in a specified (left and right) context, while a deletion operation means removing a substring of a given string from a specified (left and right) context. A finite set of insertion-deletion rules, together with a set of axioms provide a language generating device: starting from the set of initial strings and iterating insertion-deletion operations as defined by the given rules, one obtains a language.

Even in their basic variants, insertion-deletion systems are able to characterize the recursively enumerable languages. Moreover, as it was shown in [15], the context dependency may be replaced by

insertion and deletion of strings of sufficient length, in a context-free manner. If the length is not suffi-
cient (less or equal to two) then such systems are not able to generate more than the recursive languages
and a characterization of them was shown in [23].

Similar investigations were continued in [16, 11, 12] on insertion-deletion systems with one-sided
contexts, i.e., where the context dependency is present only from the left or only from the right side of
all insertion and deletion rules. The papers cited above give several computational completeness results
depending on the size of insertion and deletion rules. We recall the interesting fact that some combi-
nations are not leading to computational completeness, i.e., there are recursively enumerable languages
that cannot be generated by such devices.

Like in the case of context-free rewriting, it is possible to consider a graph-controlled variant of
insertion-deletion systems. Thus the rules cannot be applied at any time, as their applicability depends
on the current "state", changed by a rule application. There are several equivalent definitions of a graph-
controlled application, we consider one of them where rules are grouped in *components* (communication
graph nodes) and at each step one of the rules from the current component is chosen non-deterministically
and applied, at the same time changing the current component. Such a formalization is rather similar to
the definition of insertion-deletion P systems [18], however it is even simpler and more natural.

This article focuses on one-sided graph-controlled insertion-deletion systems where at most two sym-
bols may be present in the description of insertion and deletion rules. This correspond to systems of size
$(1,1,0;1,1,0)$, $(1,1,0;1,0,1)$, $(1,1,0;2,0,0)$, and $(2,0,0;1,1,0)$, where the first three numbers repre-
sent the maximal size of the inserted string and the maximal size of the left and right contexts, while the
last three numbers represent the same information, but for deletion rules. It is known that such systems
are not computationally complete [13], while the corresponding P systems variants are computationally
complete, which results are achieved with five components. In this article we give a simpler definition of
the concept of graph-controlled insertion-deletion systems and we show that computational completeness
can already be achieved by using a control graph with only four nodes (components).

2 Definitions

We do not present the usual definitions concerning standard concepts of the theory of formal languages
and we only refer to textbooks such as [20] for more details.

The empty string is denoted by λ. For the interval of natural numbers from k to m we write $[k..m]$.

In the following, we will use special variants of the *Geffert* normal form for type-0 grammars (see [5]
for more details).

A grammar $G = (N, T, P, S)$ is said to be in *Geffert normal form* [5] if $N = \{S, A, B, C, D\}$ and P only
contains context-free rules of the forms $S \to uSv$ with $u \in \{A, C\}^*$ and $v \in \{B, D\}^*$ as well as $S \to x$ with
$x \in (T \cup \{A, B, C, D\})^*$ and two (non-context-free) erasing rules $AB \to \lambda$ and $CD \to \lambda$.

We remark that we can easily transform the linear rules from the Geffert normal form into a set of
left-linear and right-linear rules (by increasing the number of non-terminal symbols, e.g., see [18]). More
precisely, we say that a grammar $G = (N, T, P, S)$ with $N = N' \cup N''$, $S, S' \in N'$, and $N'' = \{A, B, C, D\}$,
is in the *special Geffert normal form* if, besides the two erasing rules $AB \to \lambda$ and $CD \to \lambda$, it only has
context-free rules of the following forms:

$$X \rightarrow bY, \quad X, Y \in N', b \in T \cup N'',$$
$$X \rightarrow Yb, \quad X, Y \in N', b \in T \cup N'',$$
$$S' \rightarrow \lambda.$$

Moreover, we may even assume that, except for the rules of the forms $X \rightarrow Sb$ and $X \rightarrow S'b$, for the first two types of rules it holds that the right-hand side is unique, i.e., for any two rules $X \rightarrow w$ and $U \rightarrow w$ in P we have $U = X$.

The computation in a grammar in the special Geffert normal form is done in two stages. During the first stage, only context-free rules are applied. During the second stage, only the erasing rules $AB \rightarrow \lambda$ and $CD \rightarrow \lambda$ are applied. These two erasing rules are not applicable during the first stage as long as the left and the right part of the current string are still separated by S (or S') as all the symbols A and C are generated on the left side of these middle symbols and the corresponding symbols B and D are generated on the right side. The transition between stages is done by the rule $S' \rightarrow \lambda$. We remark that all these features of a grammar in the special Geffert normal form are immediate consequences of the proofs given in [5].

2.1 Insertion-deletion systems

An *insertion-deletion system* is a construct $ID = (V, T, A, I, D)$, where V is an alphabet; $T \subseteq V$ is the set of *terminal* symbols (in contrast, those of $V - T$ are called *non-terminal* symbols); A is a finite language over V, the strings in A are the *axioms*; I, D are finite sets of triples of the form (u, α, v), where u, α ($\alpha \neq \lambda$), and v are strings over V. The triples in I are *insertion rules*, and those in D are *deletion rules*. An insertion rule $(u, \alpha, v) \in I$ indicates that the string α can be inserted between u and v, while a deletion rule $(u, \alpha, v) \in D$ indicates that α can be removed from between the context u and v. Stated in another way, $(u, \alpha, v) \in I$ corresponds to the rewriting rule $uv \rightarrow u\alpha v$, and $(u, \alpha, v) \in D$ corresponds to the rewriting rule $u\alpha v \rightarrow uv$. By $\Longrightarrow_{ins}$ we denote the relation defined by the insertion rules (formally, $x \Longrightarrow_{ins} y$ if and only if $x = x_1 uv x_2, y = x_1 u\alpha v x_2$, for some $(u, \alpha, v) \in I$ and $x_1, x_2 \in V^*$), and by $\Longrightarrow_{del}$ the relation defined by the deletion rules (formally, $x \Longrightarrow_{del} y$ if and only if $x = x_1 u\alpha v x_2, y = x_1 uv x_2$, for some $(u, \alpha, v) \in D$ and $x_1, x_2 \in V^*$). By $\Longrightarrow$ we refer to any of the relations $\Longrightarrow_{ins}, \Longrightarrow_{del}$, and by $\Longrightarrow^*$ we denote the reflexive and transitive closure of $\Longrightarrow$.

The language generated by ID is defined by

$$L(ID) = \{ w \in T^* \mid x \Longrightarrow^* w \text{ for some } x \in A \}.$$

The complexity of an insertion-deletion system $ID = (V, T, A, I, D)$ is described by the vector

$$(n, m, m'; p, q, q')$$

called *size*, where

$$n = \max\{|\alpha| \mid (u, \alpha, v) \in I\}, \quad p = \max\{|\alpha| \mid (u, \alpha, v) \in D\},$$
$$m = \max\{|u| \mid (u, \alpha, v) \in I\}, \quad q = \max\{|u| \mid (u, \alpha, v) \in D\},$$
$$m' = \max\{|v| \mid (u, \alpha, v) \in I\}, \quad q' = \max\{|v| \mid (u, \alpha, v) \in D\}.$$

By $INS_n^{m,m'} DEL_p^{q,q'}$ we denote the families of insertion-deletion systems having size $(n, m, m'; p, q, q')$.

If one of the parameters n, m, m', p, q, q' is not specified, then instead we write the symbol $*$. In particular, $INS_*^{0,0} DEL_*^{0,0}$ denotes the family of languages generated by *context-free insertion-deletion systems*. If one of numbers from the pairs m, m' and/or q, q' is equal to zero (while the other one is not), then we say that the corresponding families have a one-sided context. Finally we remark that the rules from I and D can be put together into one set of rules R by writing $(u, \alpha, v)_{ins}$ for $(u, \alpha, v) \in I$ and $(u, \alpha, v)_{del}$ for $(u, \alpha, v) \in D$.

2.2 Graph-controlled insertion-deletion systems

Like context-free grammars, insertion-deletion systems may be extended by adding some additional controls. We discuss here the adaptation of the idea of programmed and graph-controlled grammars for insertion-deletion systems.

A *graph-controlled insertion-deletion system* is a construct

$$\Pi = (V, T, A, H, I_0, I_f, R) \text{ where}$$

- V is a finite alphabet,
- $T \subseteq V$ is the *terminal alphabet*,
- $A \subseteq V^*$ is a finite set of *axioms*,
- H is a set of labels associated (in a one-to-one manner) to the rules in R,
- $I_0 \subseteq H$ is the set of *initial labels*,
- $I_f \subseteq H$ is the set of *final labels*, and
- R is a finite set of rules of the form $l : (r, E)$ where r is an insertion or deletion rule over V and $E \subseteq H$.

As is common for graph controlled systems, a configuration of Π is represented by a pair (i, w), where i is the label of the rule to be applied and w is the current string. A transition $(i, w) \Rightarrow (j, w')$ is performed if there is a rule $l : ((u, \alpha, v)_t, E)$ in R such that $w \Longrightarrow_t w'$ by the insertion/deletion rule $(u, \alpha, v)_t, t \in \{ins, del\}$, and $j \in E$. The result of the computation consists of all terminal strings reaching a final label from an axiom and the initial label, i.e.,

$$L(\Pi) = \{w \in T^* \mid (i_0, w') \Rightarrow^* (i_f, w) \text{ for some } w' \in A, \ i_0 \in I_0, \ i_f \in I_f\}.$$

We may use another rather similar definiton for a graph-controlled insertion-deletion system, thereby assigning groups of rules to *components* of the system:

A *graph-controlled insertion-deletion system with k components* is a construct

$$\Pi = (k, V, T, A, H, i_0, i_f, R) \text{ where}$$

- k is the number of components,
- V, T, A, H are defined as for graph-controlled insertion-deletion systems,
- $i_0 \in [1..k]$ is the initial component,
- $i_f \in [1..k]$ is the final component, and
- R is a finite set of rules of the form $l : (i, r, j)$ where r is an insertion or deletion rule over V and $i, j \in [1..k]$.

The set of rules R may be divided into sets R_i assigned to the *components* $i \in [1..k]$, i.e., $R_i = \{l : (r, j) \mid l : (i, r, j) \in R\}$; in a rule $l : (i, r, j)$, the number j specifies the *target component* where the string is sent from component i after the application of the insertion or deletion rule r. A configuration of Π is represented by a pair (i, w), where i is the number of the *current* component (initially i_0) and w is the current string. We also say that w is *situated* in component i. A transition $(i, w) \Rightarrow (j, w')$ is performed as follows: first, a rule $l : (r, j)$ from component i (from the set R_i) is chosen in a non-deterministic way, the rule r is applied, and the string is moved to component j; hence, the new set from which the next rule to be applied will be chosen is R_j. More formally, $(i, w) \Rightarrow (j, w')$ if there is $l : ((u, \alpha, v)_t, j) \in R_i$ such that $w \Longrightarrow_t w'$ by the rule $(u, \alpha, v)_t$; we also write $(i, w) \Rightarrow_l (j, w')$ in this case. The result of the computation consists of all terminal strings situated in component i_f reachable from the axiom and the initial component, i.e.,

$$L(\Pi) = \{w \in T^* \mid (i_0, w') \Rightarrow^* (i_f, w) \text{ for some } w' \in A\}.$$

Is is not difficult to see that graph-controlled insertion-deletion systems with k components are a special case of graph-controlled insertion-deletion systems. Without going into technical details, we just give the main ideas how to obtain a graph-controlled insertion-deletion system from a graph-controlled insertion-deletion system with k components: for every $l : ((u, \alpha, v)_t, j) \in R_i$ we take a rule $l : (i, (u, \alpha, v)_t, Lab(R_j))$ into R where $Lab(R_j)$ denotes the set of labels for the rules in R_j; moreover, we take $I_0 = Lab(R_{i_0})$ and $I_f = Lab(R_{i_f})$. Finally, we remark that the labels in a graph-controlled insertion-deletion system with k components may even be omitted, but they are useful for specific proof constructions. On the other hand, by a standard powerset construction for the labels (as used for the determinization of non-deterministic finite automata) we can easily prove the converse inclusion, i.e., that for any graph-controlled insertion-deletion system we can construct an equivalent graph-controlled insertion-deletion system with k components.

We define the *communication graph* of a graph-controlled insertion-deletion system with k components to be the graph with nodes $1, \ldots, k$ having an edge from node i to node j if and only if there exists a rule $l : ((u, \alpha, v)_t, j) \in R_i$. In [18], 5.5, special emphasis is laid on graph-controlled insertion-deletion systems with k components whose communication graph has a tree structure, as we observe that the presentation of graph-controlled insertion-deletion systems with k components given above in the case of a tree structure is rather similar to the definition of insertion-deletion P systems as given in [18]; the main differences are that in P systems the final component i_f contains no rules and corresponds with the root of the communication tree; on the other hand, in graph-controlled insertion-deletion system with k components, each of the axioms can only be situated in the initial component i_0, whereas in P systems we may situate each axiom in various different components.

Throughout the rest of this paper we shall only use the notion of graph-controlled insertion-deletion systems with k components, as they are easier to handle and sufficient to establish computational completeness in the proofs of our main results presented in the succeeding section. By $GCL_k(ins_n^{m,m'}, del_p^{q,q'})$ we denote the family of languages $L(\Pi)$ generated by graph-controlled insertion-deletion systems with at most k components and insertion and deletion rules of size at most $(n, m, m'; p, q, q')$. We replace k by $*$ if k is not fixed. The letter "G" is replaced by the letter "T" to denote classes whose communication graph has a *tree structure*. Some results for the families $TCL_k(ins_n^{m,m'}, del_p^{q,q'})$ can directly be derived from the results presented in [13, 18] for the corresponding families of insertion-deletion P systems $ELSP_k(ins_n^{m,m'}, del_p^{q,q'})$, yet the results we present in the succeeding section either reduce the number of components for systems with an underlying tree structure or else take advantage of the arbitrary structure of the underlying communication graph thus obtaining computational completeness for new restricted

variants of insertion and deletion rules.

3   Main results

For all the variants of insertion and deletion rules considered in this section, we know that the basic variants without using control graphs cannot achieve computational completeness (see [13], [16]). The computational completeness results from this section are based on simulations of derivations of a grammar in the special Geffert normal form. These simulations associate a group of insertion and deletion rules to each of the right- or left-linear rules $X \to bY$ and $X \to Yb$. The same holds for (non-context-free) erasing rules $AB \to \lambda$ and $CD \to \lambda$. We remark that during the derivation of a grammar in the special Geffert normal form, any sentential form contains at most one non-terminal symbol from N'.

   We start with the following theorem where we even obtain a linear tree structure for the underlying communication graph.

Theorem 1. $TCL_4(ins_1^{1,0}, del_2^{0,0}) = RE$.

Proof. Consider a type-0 grammar $G = (N, T, P, S)$ in the special Geffert normal form. We construct a graph-controlled insertion-deletion system

$$\Pi = (4, V, T, \{S\}, H, 1, 1, R)$$

that simulates G as follows. The rules from P are supposed to be labeled in a one-to-one manner with labels from the set $[1..|P|]$. The alphabet of Π is $V = N \cup T \cup \{p, p' \mid p : X \to \alpha \in P\}$. The set of rules R of Π is defined as follows:

   For any rule $p : X \to bY$ we take the following insertion and deletion rules into R:

$$p.1.1 : (1, (X, p, \lambda)_{ins}, 2)$$
$$p.2.1 : (2, (p, Y, \lambda)_{ins}, 3) \qquad\qquad p.2.2 : (2, (\lambda, p', \lambda)_{del}, 1)$$
$$p.3.1 : (3, (p, p', \lambda)_{ins}, 4) \qquad\qquad p.3.2 : (3, (p', b, \lambda)_{ins}, 2)$$
$$p.4.1 : (4, (\lambda, Xp, \lambda)_{del}, 3)$$

   For any rule $p : X \to Yb$ we take the following insertion and deletion rules into R:

$$p.1.1 : (1, (X, p, \lambda)_{ins}, 2)$$
$$p.2.1 : (2, (p, b, \lambda)_{ins}, 3) \qquad\qquad p.2.2 : (2, (\lambda, p', \lambda)_{del}, 1)$$
$$p.3.1 : (3, (p, p', \lambda)_{ins}, 4) \qquad\qquad p.3.2 : (3, (p', Y, \lambda)_{ins}, 2)$$
$$p.4.1 : (4, (\lambda, Xp, \lambda)_{del}, 3)$$

   For simulating the erasing productions $AB \to \lambda$ and $CD \to \lambda$ as well as $S' \to \lambda$ we add the rules $(1, (\lambda, AB, \lambda)_{del}, 1)$ and $(1, (\lambda, CD, \lambda)_{del}, 1)$ as well as $(1, (\lambda, S', \lambda)_{del}, 1)$ to R..

   We claim that $L(\Pi) = L(G)$. We start by proving the inclusion $L(G) \subseteq L(\Pi)$. Let $S \Longrightarrow^* uXv \Longrightarrow ubYv \Longrightarrow^* w$ be a derivation of a string $w \in L(G)$. We show that Π correctly simulates the application of the rule $p : X \to bY$. Consider the string uXv in component 1. Then, there is only one possible sequence of applications of rules in Π:

$$(1, uXv) \Rightarrow_{p.1.1} (2, uXpv) \Rightarrow_{p.2.1} (3, uXpYv) \Rightarrow_{p.3.1} (4, uXpp'Yv)$$
$$\Rightarrow_{p.4.1} (3, up'Yv) \Rightarrow_{p.3.2} (2, up'bYv) \Rightarrow_{p.2.2} (1, ubYv).$$

In a similar way the rules $X \to Yb$ are simulated:

$$(1,uXv) \Rightarrow_{p.1.1} (2,uXpv) \Rightarrow_{p.2.1} (3,uXpbv) \Rightarrow_{p.3.1} (4,uXpp'bv)$$
$$\Rightarrow_{p.4.1} (3,up'bv) \Rightarrow_{p.3.2} (2,up'Ybv) \Rightarrow_{p.2.2} (1,uYbv).$$

The rules $AB \to \lambda$ and $CD \to \lambda$ as well as $S' \to \lambda$ are directly simulated by the corresponding deletion rules $(1,(\lambda,AB,\lambda)_{del},1)$ and $(1,(\lambda,CD,\lambda)_{del},1)$ as well as $(1,(\lambda,S',\lambda)_{del},1)$ from R in component 1.

Hence, observing that initially S is present in component 1, and by applying the induction argument we obtain that there is a derivation $(1,S) \Longrightarrow^* (1,w)$ in Π. Thus, we conclude $L(G) \subseteq L(\Pi)$. For the converse inclusion, it is sufficient to observe that any computation in Π can only be performed by applying the group of rules corresponding to a production of G. Thus, for any derivation in Π a corresponding derivation in G can be obtained.

Finally we observe that the rules in R induce an even linear structure for the communication graph, which for short can be represented as follows:

$$\circlearrowleft 1 \rightleftarrows 2 \rightleftarrows 3 \rightleftarrows 4$$

This observation concludes the proof. $\hspace{2cm} \square$

The next theorem uses one-sided contextual deletion rules.

Theorem 2. $GCL_4(ins_1^{1,0}, del_1^{1,0}) = RE$.

Proof. Consider a type-0 grammar $G = (N,T,P,S)$ in the special Geffert normal form. We construct a graph-controlled insertion-deletion system

$$\Pi = (4,V,T,\{S\},H,1,1,R)$$

that simulates G as follows. The rules from P are supposed to be labeled in a one-to-one manner with labels from the set $[1..|P|]$. The alphabet of Π is $V = N \cup T \cup \{K,K'\} \cup \{p \mid p : X \to \alpha \in P\}$. The set of rules R of Π is defined as follows.

For any rule $p : X \to bY$ we take the following insertion and deletion rules into R:

$$p.1.1 : (1,(\lambda,p,\lambda)_{ins},2)$$
$$p.2.1 : (2,(p,X,\lambda)_{del},3) \hspace{2cm} p.2.2 : (2,(\lambda,p,\lambda)_{del},1)$$
$$p.3.1 : (3,(p,Y,\lambda)_{ins},4)$$
$$p.4.1 : (4,(p,b,\lambda)_{ins},2)$$

For any rule $p : X \to Yb$ we take the following insertion and deletion rules into R:

$$p.1.1 : (1,(\lambda,p,\lambda)_{ins},2)$$
$$p.2.1 : (2,(p,X,\lambda)_{del},3) \hspace{2cm} p.2.2 : (2,(\lambda,p,\lambda)_{del},1)$$
$$p.3.1 : (3,(p,b,\lambda)_{ins},4)$$
$$p.4.1 : (4,(p,Y,\lambda)_{ins},2)$$

For the erasing production $S' \to \lambda$ we have to add the rule $(1,(\lambda,S',\lambda)_{del},1)$, and for the erasing productions $AB \to \lambda$ and $CD \to \lambda$ we take the following rules into R:

$$k.1.1 : (1,(\lambda,K,\lambda)_{ins},2) \qquad k.1.2 : \left(1,(\lambda,K',\lambda)_{ins},2\right)$$
$$k.2.1 : (2,(K,A,\lambda)_{del},3) \qquad k.2.2 : \left(2,(K',C,\lambda)_{del},3\right)$$
$$k.2.3 : (2,(\lambda,K,\lambda)_{del},1) \qquad k.2.4 : \left(2,(\lambda,K',\lambda)_{del},1\right)$$
$$k.3.1 : (3,(K,B,\lambda)_{del},2) \qquad k.3.2 : \left(3,(K',D,\lambda)_{del},2\right)$$

The simulation of a rule $p : X \to bY$ of G is performed as follows. Let the current sentential form be uXv. There are several possibilities here. First, the symbol p is inserted in a context-free manner anywhere in the string by rule $p.1.1$. After that, either rule $p.2.2$ is applicable, or, if p was inserted before X, rule $p.2.1$ is applicable. In the first case the string remains unchanged: uXv. We remark that this is also the only evolution if a symbol $q \neq p$ is inserted. In the second case, there is only one possible further evolution, yielding the desired result $ubYv$ in component 1:
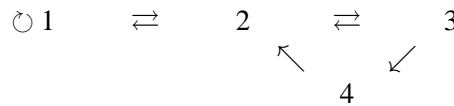
$$(2,upXv) \Rightarrow_{p.2.1} (3,upv) \Rightarrow_{p.3.1} (4,upYv) \Rightarrow_{p.4.1} (2,upbYv) \Rightarrow_{p.2.2} (1,ubYv)$$

In a similar way the rules $X \to Yb$ are simulated:

$$(2,upXv) \Rightarrow_{p.2.1} (3,upv) \Rightarrow_{p.3.1} (4,upbv) \Rightarrow_{p.4.1} (2,upYbv) \Rightarrow_{p.2.2} (1,uYbv)$$

Now consider the simulation of the rule $AB \to \lambda$ (the case of the rule $CD \to \lambda$ is treated in an analogous way). Suppose that K is inserted in a context-free manner in string u by rule $k.1.1$ and that we obtain a string $u'Ku''$ in component 2. After that, either rule $k.2.1$ is applicable if K was inserted before A, i.e., $u'Ku'' = u'KAu'''$, and we obtain the string $u'Ku'''$ in component 3, or the string u remains unchanged and returns to component 1 by applying rule $k.2.3$. In the first case, if the first letter of u''' is not equal to B, the evolution of this string is stopped. Otherwise, if $u''' = Bu^{iv}$, rule $k.3.1$ is applied and the string $u'Ku^{iv}$ is obtained in component 2. Now the computation may be continued in the same manner and K either eliminates another couple of symbols AB if this is possible, or the string appears in component 1 without K and then is ready for new evolutions.

Now in order to complete the proof, we observe that the only sequences of rules leading to a terminal derivation in Π correspond to the groups of rules as defined above. Hence, a derivation in G can be reconstructed from a derivation in Π. Finally we remark that in contrast to the preceding theorem, the communication graph has no tree structure, yet instead looks like as follows:

These observations conclude the proof. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad$ $\square$

The result elaborated above also holds if the contexts for insertion and deletion rules are on different sides.

Theorem 3. $GCL_4(ins_1^{1,0}, del_1^{0,1}) = RE$.

Proof. We modify the proof of Theorem 2 as follows. We replace the rules $p.2.1$ by the corresponding rules $(2,(\lambda,X,p)_{del},3)$ and the rules $k.2.1$, $k.2.2$, $k.3.1$, and $k.3.2$ by their symmetric versions. In this case we get a derivation which differs from the derivation of the previous theorem only by the position of the deleting symbol, which is inserted in a context-free manner. Hence, the derivations are equivalent and lead to the same result.   □

Finally, we prove that a similar result also holds in the case of context-free insertions.

Theorem 4. $GCL_4(ins_2^{0,0}, del_1^{1,0}) = RE$.

Proof. Consider a type-0 grammar $G = (N,T,P,S)$ in the special Geffert normal form. We construct a graph-controlled insertion-deletion system

$$\Pi = (4,V,T,\{S\},H,1,1,R)$$

that simulates G as follows. The rules from P are supposed to be labeled in a one-to-one manner with labels from the set $[1..|P|]$. The alphabet of Π is $V = N \cup T \cup \{K,K'\} \cup \{p \mid p : X \to \alpha \in P\}$. The set of rules R of Π is defined as follows.

For any rule $p : X \to bY$ we take the following insertion and deletion rules to R (we stress that only one symbol Y is present in the developing string):

$$p.1.1 : (1,(\lambda,bp,\lambda)_{ins},2)$$
$$p.2.1 : (2,(p,X,\lambda)_{del},3) \qquad\qquad p.2.2 : (2,(Y,p,\lambda)_{del},1),$$
$$p.3.1 : (3,(\lambda,Y,\lambda)_{ins},2)$$

For any rule $p : X \to Yb$ we take the following insertion and deletion rules into R:

$$p.1.1 : (1,(\lambda,Yp,\lambda)_{ins},2)$$
$$p.2.1 : (2,(p,X,\lambda)_{del},3) \qquad\qquad p.2.2 : (2,(b,p,\lambda)_{del},1)$$
$$p.3.1 : (3,(\lambda,b,\lambda)_{ins},2)$$

For the erasing production $S' \to \lambda$ we have to add the rule $(1,(\lambda,S',\lambda)_{del},1)$; the erasing rules $AB \to \lambda$ and $CD \to \lambda$ are simulated by the following rules in R:

$$k.1.1 : (1,(\lambda,K,\lambda)_{ins},2) \qquad\qquad k.2.1 : \left(1,(\lambda,K',\lambda)_{ins},2\right)$$
$$k.2.1 : (2,(K,A,\lambda)_{del},3) \qquad\qquad k.2.2 : \left(2,(K',C,\lambda)_{del},3\right)$$
$$k.3.1 : (3,(K,B,\lambda)_{del},4) \qquad\qquad k.3.2 : \left(3,(K',D,\lambda)_{del},4\right)$$
$$k.4.1 : (4,(\lambda,K,\lambda)_{del},1) \qquad\qquad k.4.2 : \left(4,(\lambda,K',\lambda)_{del},1\right)$$

The simulation of a rule $X \to bY$ is performed as follows:

$$(1,uXv) \Rightarrow_{p.1.1} (2,ubpXv) \Rightarrow_{p.2.1} (3,ubpv) \Rightarrow_{p.3.1} (2,ubYpv) \Rightarrow_{p.2.2} (1,ubYv)$$

Since the rules $p.1.1$ and $p.3.1$ perform a context-free insertion, the corresponding string can be inserted anywhere. However, if it is not inserted at the right position, then the computation is immediately blocked, because the corresponding deletion cannot be performed.

The simulation of a rule $X \to Yb$ is performed in a similar way:

$$(1, uXv) \Rightarrow_{p.1.1} (2, uYpXv) \Rightarrow_{p.2.1} (3, uYpv) \Rightarrow_{p.3.1} (2, uYbpv) \Rightarrow_{p.2.2} (1, uYbv)$$

Observe that $p.2.2$ cannot be used instead of $p.2.1$ because $Y \neq b$.

The erasing rule $AB \to \lambda$ is simulated as follows (the construction for $CD \to \lambda$ is very similar, using K' instead of K, so we omit it here):

$$(1, uABv) \Rightarrow_{k.1.1} (2, uKABv) \Rightarrow_{k.2.1} (3, uKBv) \Rightarrow_{k.3.1} (4, uKv) \Rightarrow_{k.4.1} (1, uv)$$

The communication graph is identical to the graph in the proof of Theorem 1:

$$\circlearrowleft 1 \rightleftarrows 2 \rightleftarrows 3 \rightleftarrows 4$$

Finally, we remark that only simulations of rules from G as described above may be part of derivations in Π yielding a terminal string; hence, we conclude $L(\Pi) = L(G)$. $\qquad\square$

4 Conclusions

In this article we have investigated the application of the mechanism of a control graph to the operations of insertion and deletion. We gave a clear definition of the corresponding systems, which is simpler than the one obtained by using P systems. We investigated the case of systems with insertion and deletion rules of size $(1,1,0;1,1,0)$, $(1,1,0;1,0,1)$, $(1,1,0;2,0,0)$ and $(2,0,0;1,1,0)$ and we have shown that the corresponding graph-controlled insertion-deletion systems are computationally complete with only four components, i.e., with the underlying communication graph containing only four nodes. The case of graph-controlled systems having rules of size $(2,0,0;2,0,0)$ is investigated in [13], where it is shown that such systems are not computationally complete.

We suggest two directions for the future research. The first one deals with the number of components needed to achieve computational completeness. The natural question is if it is possible to obtain similar results with only three components. The second direction is inspired from the area of P systems. We propose to further investigate systems where the communication graph has a tree structure as in Theorem 1. The only known results so far are to be found in [13], but there five nodes were used. Hence, the challenge remains to decrease these numbers of components.

References

[1] R. Benne (1993): *RNA Editing: The Alteration of Protein Coding Sequences of RNA*. Ellis Horwood, Chichester, West Sussex.

[2] Franziska Biegler, Michael J. Burrell & Mark Daley (2007): *Regulated RNA Rewriting: Modelling RNA Editing with Guided Insertion*. Theor. Comput. Sci. 387(2), pp. 103 – 112. Descriptional Complexity of Formal Systems.

[3] Mark Daley, Lila Kari, Greg Gloor & Rani Siromoney (1999): *Circular Contextual Insertions/Deletions with Applications to Biomolecular Computation*. In: *SPIRE/CRIWG*, pp. 47–54.

[4] B.S. Galiukschov (1981): *Semicontextual Grammars*. Matem. Logica i Matem. Lingvistika, pp. 38–50 Tallin University, (in Russian).

[5] Viliam Geffert (1991): *Normal forms for phrase-structure grammars*. ITA 25, pp. 473–498.

[6] D. Haussler (1982): *Insertion and Iterated Insertion as Operations on Formal Languages*. Ph.D. thesis, Univ. of Colorado at Boulder.

[7] D. Haussler (1983): *Insertion Languages*. Information Sciences 31(1), pp. 77–89.

[8] L. Kari, Gh. Păun, G. Thierrin & S. Yu (1997): *At the Crossroads of DNA Computing and Formal Languages: Characterizing RE Using Insertion-Deletion Systems*. In: *Proc. of 3rd DIMACS Workshop on DNA Based Computing*, Philadelphia, pp. 318–333.

[9] Lila Kari (1991): *On Insertion and Deletion in Formal Languages*. Ph.D. thesis, University of Turku.

[10] S. C. Kleene (1956): *Representation of Events in Nerve Nets and Finite Automata*. In: Claude Shannon & John McCarthy, editors: *Automata Studies*, Princeton University Press, Princeton, NJ, pp. 3–41.

[11] Alexander Krassovitskiy, Yurii Rogozhin & Sergey Verlan (2008): *Further Results on Insertion-Deletion Systems with One-Sided Contexts*. In: Carlos Martín-Vide, Friedrich Otto & Henning Fernau, editors: *Language and Automata Theory and Applications, Second International Conference, LATA 2008, Tarragona, Spain, March 13-19, 2008. Revised Papers, Lecture Notes in Computer Science* 5196, Springer, pp. 333–344.

[12] Alexander Krassovitskiy, Yurii Rogozhin & Sergey Verlan (2009): *Computational Power of P Systems with Small Size Insertion and Deletion Rules*. In: Turlough Neary, Damien Woods, Anthony Karel Seda & Niall Murphy, editors: *Proceedings International Workshop on The Complexity of Simple Programs, Cork, Ireland, 6-7th December 2008, EPTCS* 1, pp. 108–117.

[13] Alexander Krassovitskiy, Yurii Rogozhin & Sergey Verlan (2010): *Computational Power of Insertion-Deletion (P) Systems with Rules of Size Two*. *Natural Computing*, 2010, (in publication).

[14] Solomon Marcus (1969): *Contextual Grammars*. *Rev. Roum. Math. Pures Appl.* 14, pp. 1525–1534.

[15] Maurice Margenstern, Gheorghe Păun, Yurii Rogozhin & Sergey Verlan (2005): *Context-Free Insertion-Deletion Systems*. *Theor. Comput. Sci.* 330(2), pp. 339–348.

[16] Artiom Matveevici, Yurii Rogozhin & Sergey Verlan (2007): *Insertion-Deletion Systems with One-Sided Contexts*. In: Jérôme Olivier Durand-Lose & Maurice Margenstern, editors: *Machines, Computations, and Universality, 5th International Conference, MCU 2007, Orléans, France, September 10-13, 2007, Proceedings, Lecture Notes in Computer Science* 4664, Springer, pp. 205–217.

[17] Gheorghe Păun (1997): *Marcus Contextual Grammars*. Kluwer Academic Publishers, Norwell, MA, USA.

[18] Gheorghe Păun (2002): *Membrane Computing. An Introduction*. Springer-Verlag.

[19] Gheorghe Păun, Grzegorz Rozenberg & Aarto Salomaa (1998): *DNA Computing: New Computing Paradigms*. Springer.

[20] Grzegorz Rozenberg & Arto Salomaa, editors (1997): *Handbook of Formal Languages*. Springer-Verlag, Berlin.

[21] W. D. Smith (1996): *DNA Computers in Vitro and in Vivo*. In: R.J. Lipton & E.B. Baum, editors: *Proceedings of DIMACS Workshop on DNA Based Computers*, DIMACS Series in Discrete Math. and Theoretical Computer Science, Amer. Math. Society, pp. 121–185.

[22] Akihiro Takahara & Takashi Yokomori (2002): *On the Computational Power of Insertion-Deletion Systems*. In: Masami Hagiya & Azuma Ohuchi, editors: *DNA Computing, 8th International Workshop on DNA Based Computers, DNA8, Sapporo, Japan, June 10-13, 2002, Revised Papers, Lecture Notes in Computer Science* 2568, Springer, pp. 269–280.

[23] Sergey Verlan (2007): *On Minimal Context-Free Insertion-Deletion Systems*. *Journal of Automata, Languages and Combinatorics* 12(1-2), pp. 317–328.